(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 1 081 592 A2

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
07.03.2001 Bulletin 2001/10

(51) Int. Cl.[7]: **G06F 9/44**

(21) Application number: **00307177.6**

(22) Date of filing: **21.08.2000**

(84) Designated Contracting States:
AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE
Designated Extension States:
AL LT LV MK RO SI

(30) Priority: **30.08.1999 US 386093**

(71) Applicant:
LUCENT TECHNOLOGIES INC.
Murray Hill, New Jersey 07974-0636 (US)

(72) Inventors:
- **Ball, Thomas J.**
  Mercier Island, Washington 98040 (US)
- **Danielsen, Peter John**
  Naperville, Illinois 60540 (US)
- **Jagadeesan, Lalita J.**
  Naperville, Illinois 60540 (US)
- **Laufer, Konstantin**
  Chicago, Illinois 60660 (US)
- **Mataga, Peter Andrew**
  Naperville, Illinois 60563 (US)
- **Rehor, Kenneth G.**
  Chicago, Illinois 60647 (US)

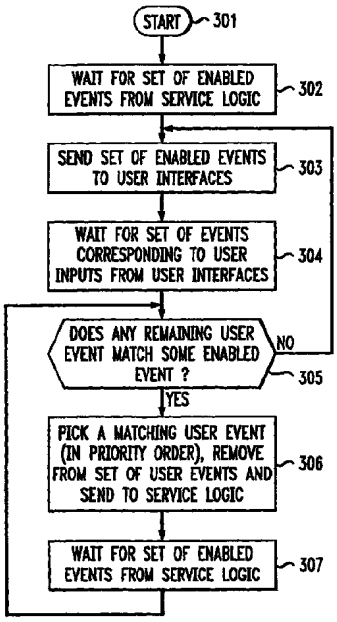
(74) Representative:
Buckley, Christopher Simon Thirsk et al
Lucent Technologies (UK) Ltd,
5 Mornington Road
Woodford Green, Essex IG8 0TU (GB)

(54) Method and apparatus for providing interactive services with multiple interfaces

(57) Interactive services with multiple interfaces are realized by employing a modular approach to their implementation. Such an approach facilitates supporting natural language understanding interaction with users through use of interfaces that at least allow the user to provide information beyond what is currently being requested by the service, and/or different ordering of inputs, and/or incomplete information, and/or correction of information, and/or the return of control to prior points in the service. This is realized, in an embodiment of the invention, by employing an interactive service logic that uses "*reactive constraint graphs*", i.e., a form of event-driven graph, in which nodes contain constraints on events, in conjunction with a service monitor. The service monitor manages the communication between the service logic and the multiple user interfaces. As such it provides a communication mechanism in the form of a so-called "look ahead" in which the user can input information beyond what is currently required by the interactive service. Specifically, in each round of user interaction, reactive constraint graphs report to the service monitor the set of events that are currently enabled. These enable events correspond to the information the interactive service is currently requesting (or ready to accept) from a user. The user interfaces collect information (perhaps beyond this set) from the user and send it to the service monitor. The service monitor then matches the received information against the information requested by the service logic, and sends the information one event at a time to the service logic, in a priority order specified by the service provider. After each event is sent to the service logic from the service monitor, the service logic reports its new set of enabled events back to the service monitor. In turn, the service monitor sends another event that matches the enabled events to the service logic. The service monitor then notifies the user interfaces to send a new collection, i.e., set, of information from the individual users only when no current information matches the enabled events. This approach advantageously allows a user to input information beyond what is currently needed by the interactive service.

EP 1 081 592 A2

FIG. 3


START
301
WAIT FOR SET OF ENABLED EVENTS FROM SERVICE LOGIC
302
SEND SET OF ENABLED EVENTS TO USER INTERFACES
303
WAIT FOR SET OF EVENTS CORRESPONDING TO USER INPUTS FROM USER INTERFACES
304
DOES ANY REMAINING USER EVENT MATCH SOME ENABLED EVENT ?
NO
305
YES
PICK A MATCHING USER EVENT (IN PRIORITY ORDER), REMOVE FROM SET OF USER EVENTS AND SEND TO SERVICE LOGIC
306
WAIT FOR SET OF ENABLED EVENTS FROM SERVICE LOGIC
307

Description

Related Application

[0001] United States patent application Serial No. (T. J. Ball et al. Case 10-4-2-1-9-7) was filed concurrently herewith.

Technical Field

[0002] This invention relates to providing interactive services and, more particularly, to interactive services with multiple interfaces.

Background of the Invention

[0003] Prior arrangements for implementing interactive services with multiple interfaces employed finite-state machines (FSM) to realize the necessary service logic. This was achieved either explicitly, using FSM tools such as ObjecTime or StateCharts, or implicitly, using programming languages/technologies such as C, C++, Perl or Microsoft Active Server Pages (ASP).

[0004] In order to support natural language understanding interaction with users, services with multiple flexible interfaces need to allow the user to provide information beyond what is currently being requested by the service, different ordering of inputs, incomplete information, correction of information and the return of control to previous points in the service. Finite-state machine based approaches, however, cannot support these desired aspects in a modular fashion. Indeed, every possible ordering of inputs needs to be explicitly specified, resulting in an exponentially increasing number of states and transitions in the FSM. Additionally, including the other desirable aspects noted above further increases the size and complexity of the FSM. Moreover, it is extremely difficult to modify the FSM; for example, the number of modifications required to the FSM to add another input can be exponential in the size of the input set.

[0005] User interaction with the prior interactive services is in so-called rounds. During each round the user must provide exactly the information requested by the service in its then current state. Specifically, in a graphical interface, the user must click on buttons corresponding to input choices; in a web-based interface, the user must click on buttons or fill in text fields; in a touch-tone telephone interface, the user must press buttons of the key pad; and in an automatic speech recognition based interface, the user must utter exactly the information that the service requires in its current state. Such arrangements are very restrictive in requiring the exactly needed information.

[0006] Prior arrangements employed to provide spoken dialog interaction are focused on a single, spoken language interface, which is limiting.

Summary Of The Invention

[0007] Problems and limitations of prior known arrangements for providing interactive services are overcome by employing a modular approach to implementing interactive services with a plurality, i.e., multiple, user interfaces. Such an approach facilitates supporting natural language understanding interaction with users through use of user interfaces that at least allow the user to provide information beyond what is currently being requested by the service, and/or different ordering of inputs from the user interfaces, and/or incomplete information from the user interfaces, and/or correction of information from the user interfaces, and/or the return of control to prior points in the service in response to a request from at least one of the plurality of user interfaces.

[0008] This is realized, in an embodiment of the invention, by employing an interactive service logic that uses "*reactive constraint graphs*", i.e., a form of event-driven graph in which nodes contain constraints on events, in conjunction with a service monitor. The service monitor manages the communication between the service logic and the multiple user interfaces. As such it provides a communication mechanism in the form of a so-called "look ahead" in which the user can input information beyond what is currently required by the interactive service.

[0009] Specifically, in each round of user interaction, reactive constraint graphs report to the service monitor the set of events that are currently enabled. These enabled events correspond to the information the interactive service is currently requesting (or ready to accept) from a user. The user interfaces collect information (perhaps beyond this set) from the user and send it to the service monitor. The service monitor then matches the received information against the information requested by the service logic, and sends the information one event at a time to the service logic, in a priority order specified by the service provider. After each event is sent to the service logic from the service monitor, the service logic reports its new set of enabled events back to the service monitor. In turn, the service monitor sends another event that matches the enabled events to the service logic. The service monitor then notifies the user interfaces to send a new collection, i.e., set, of information from the individual users only when no current information matches the enabled events. This approach advantageously allows a user to input information beyond what is currently needed by the interactive service.

Brief Description Of The Drawing

[0010]

FIG. 1 shows, in simplified block diagram form, details of an interactive system in which the inven-

tion may be advantageously employed;

FIG. 2 shows, in simplified block form, details of the SISL (Several Interfaces, Single Logic) architecture employed in the embodiment of FIG. 1;

FIG. 3 is a flow chart illustrating the steps implemented by the service monitor in accordance with the invention;

FIG. 4 is a flow diagram illustrating a reactive constraint graph for a portion of an Any-Time Teller banking service example; and

FIG. 5 illustrates the steps performed in an example execution of the AnyTime Teller including look ahead in accordance with the invention.

Detailed Description

[0011] FIG. 1 shows, in simplified block diagram form, details of an interactive system in which the invention may be advantageously employed. It should be noted that the arrangement shown in FIG. 1 is but one example of an application of the invention. Indeed, a plurality of different user interfaces and/or one more identical user interfaces may be employed as desired.

[0012] Specifically, shown in FIG. 1 is SISL (Several Interfaces, Single Logic) service unit 101; home/office computer 102 used as a customer and/or provider interface and includes automatic speech recognition including natural language understanding, if desired, that is interfaced to SISL service unit 101 via an Internet link 103; telephone 104 also used as a customer and/or provider interface that is interfaced to SISL service unit 101 via a telephone network 105 including, for example, touch-tone, i.e., multi-frequency, signaling; computer 106 used as a customer and/or provider interface, which may also include a natural language understanding, that is interfaced to SISL service unit 101 via a local area network (LAN); and ATM (automatic teller machine) used as a customer interface and, typically, is interfaced to SISL service unit 101 via a direct connect 109. A key advantage of SISL is that all user interfaces to a service share the same service logic. SISL provides a clean separation between the service logic and the software for a variety of user interfaces including but not limited to Java applets, HTML pages, speech-based natural language dialog, and telephone-based voice access. In this example, SISL is implemented using the JAVA programming language.

[0013] At the outset it is felt best to describe some of the principles employed in implementing the flexible interactive service including an embodiment of the invention. For simplicity and clarity of exposition, these principles will be presented in the context of a so-called Any-Time Teller banking service employing the invention. The Any-Time Teller is an interactive banking service. The service is login protected; customers must authenticate themselves by entering an identifier (login) and PIN, i.e., personal identification number, (password) to access the functions. As customers may have many money accounts, most functions require the customer to select the account(s) involved. Once authenticated, the customer may:

- Make a deposit.
- Make a withdrawal. The service makes sure the customer has enough money in the account, then withdraws the specified amount.
- Transfer funds between accounts. The service prompts the customer to select a source and target account, and a transfer amount, and performs the transfer if (1) the customer has enough money in the source account, and (2) transfers are permitted between the two accounts.
- Get the balance of an account. Display the balance with respect to all posted transactions.
- View the transactions of an account. Display the transactions for the selected account.
- See the last transaction. Display the last action the customer performed.

[0014] The transfer capability of the Any-Time Teller requires the collection of three events: the source account (*src*), target account (*tgt*), and dollar amount (*amt*). There are five constraints that those input events must meet before a transfer can take place:

- c1 : *src* must be a valid account
- c2 : *tgt* must be a valid account
- c3 : *amt* must be greater than zero
- c4 : *amt* must be less than or equal to the balance of *src*
- c5 : the bank must allow transfers from *src* to *tgt*

[0015] The service should check whether or not a constraint is violated as soon as possible; hence, it must be prepared to react when only a subset of the three events is present. The service must then prompt for the remaining events.

[0016] Some basic principles are now considered related to the design and implementation of interactive services with multiple and varied user interfaces. These principles are especially important in the presence of multiple interfaces including those based on spoken natural language.

- *Lesser order = greater freedom*

[0017] The source account, target account, and dol-

lar amount of the transfer capability should be independent events that can be collected in any order, where all three events are necessary to perform a transfer. By making explicit independent and dependent events, it in clear what events may be reordered without affecting the behavior of the service. This points the way to our first principle of the service specification:

- Principle 1: Specify the service logic as a set of constraints on events and introduce a constraint between two events only when absolutely necessary for the correct functioning of a service. That is, the service logic should be able to accept independent input events in different orders.

• *Eliminate errors only*

[0018] It is often desirable for a service to respond as soon as possible with error conditions about user input. Since a constraint on the input events may refer to any arbitrary subset of those events, it is desirable that the service logic should be able to accept arbitrary subsets of events at any time.

- Principle 2: The service logic should accept an incomplete input, i.e., subsets of the universe of possible events.

• *I take that back!*

[0019] Unfortunately, humans often change their mind and/or make mistakes. Whenever possible, services must accommodate these shortcomings of our species, providing a capability to correct information or back out of a transaction. This leads to our third principle:

- Principle 3: The service logic should allow the user to back up to correct or update previously submitted information at any time.

A closely related principle is:

- Principle 4: The service logic should allow the user to back up to previous points in the service.

Ready or not?

[0020] Services that obey the above principles generalize from linear user interactions to potentially allow multiple points of interaction to be enabled at a given instant. This information serves as an abstraction of the current control point of the service, and can be handled in a different manner by different user interfaces. For example, in automatic speech recognition interfaces, the information about currently enabled events is used by the user interface in two ways:

- to appropriately prompt the user for information, thus compensating for the lack of visual cues; and

- to effectively parse the information provided by the user.

[0021] A user interface need not respond to all currently enabled events of the service. Thus, different user interfaces can formulate different queries to the user even though the control point in the underlying service logic, as revealed by the current set of enabled events, is the same. The decoupling that we are seeking between the user interface and the service logic, therefore points the way to our last principle of the service specification:

- Principle 5: At any point in the service, the service logic must automatically report to the user interfaces all currently enabled events, user prompts, help, and ways to revert back to previous points in the service.

[0022] User interfaces have two main responsibilities with respect to the SISL architecture that reflect the two-way information flow between the user interface and service logic:

- Based on the events received from the service logic, via the service monitor, prompt the user to provide the appropriate information and respond if the user requests help.

- Collect the information from the user and transform the information into events to be sent to the service logic, via the service monitor.

[0023] Indeed, any user interface (UI) that performs these functions can be employed in conjunction with an SISL service logic. Additionally, SISL provides a convenient framework for designing and implementing web-based, applet-based, automatic speech recognition-based and telephone voice-based interfaces. To implement such user interfaces, the UI designer need only specify two functions corresponding to the prompt and help mechanism. For automatic speech recognition interfaces, a set of speech grammars, i.e., the input to a speech recognition engine that permits it to efficiently and effectively recognize spoken input, together with a third function that specifies which grammars to enable is also required.

[0024] The required functions are:

- A *prompt* function that generates the string to be given as the prompt to the user. The SISL infrastructure automatically causes automatic speech recognition-based interfaces to speak the prompt string. Web-based interfaces automatically display the prompt string, as well as, radio buttons corre-

sponding to the possible transaction choices. For the other prompt events, text fields are automatically displayed, while submit buttons are automatically displayed for enabled events that allow the user to return to earlier points in the service.

- A *help* function that generates the string to be given as the prompt to the user.

- A *grammar* function that enables the correct set of grammar rules; this function is only needed for automatic speech recognition-based interfaces.

[0025] From prescribed functions and grammars, the SISL infrastructure automatically coordinates the collection and event transformation mechanisms, and integrates the user interface with the service logic and the service monitor. For automatic speech recognition-based interfaces, the SISL infrastructure automatically generates a desktop interface based on JAVA Speech API. To enable telephone-based voice access to the service, SISL automatically generates VoxML pages, which specify the voice dialog to be carried out on a telephony platform. For Web-based interfaces, the SISL infrastructure automatically generates HTML (Hypertext Markup Language) pages. It is noted that SISL provides a mechanism for the UI designer to customize the look and feel of the interface.

[0026] FIG. 2 shows, in simplified block form, details of the SISL (Several Interfaces, Single Logic) architecture employed in the embodiment of FIG. 1. The SISL architecture together with the event communication protocol provides modularity between the service logic 201 and user interfaces 204. In particular, two features of the event communication protocol allow service logic 201 to function completely without knowledge of the specifics of the individual user interfaces 204. These features are: (1) events are multicast from service logic 201 via service monitor 202 to user interfaces 204 and, consequently, service logic 201 does not need to know the destinations of these events; and (2) the source of the events from the user interfaces 204 is not recorded and, consequently, the service logic 201 does not know which one or more of user interfaces 204 sent the events. Service monitor 202 is responsible for maintaining this communication protocol. This modularity allows service providers to provide interchangeable user interfaces 204, or add new ones, to a single consistent source of service logic and data.

[0027] Specifically, shown in FIG. 2 are service logic unit 201, service monitor unit 202 and user interfaces 204-1 through 204-N. The key principle underlying SISL is that all user interfaces 204 to a service share a single service logic 201. All communications between the service logic 201 and its multiple user interfaces 204 are through events, via a service monitor 202. Events from the service logic 201 are broadcast to the user interfaces 204 via 203 to the service monitor 202 and, then, via 205 as a set of enabled events to the user interfaces 204. At the outset of the service, for example the Any-Time Teller banking service, each user interface 204 registers with the service monitor 202 to indicate which events it is interested in receiving. After collecting information from the user, the user interfaces 204 send events to the service monitor 202 via bi-directional links 205; the service monitor 202 does not record the source of these events. The service monitor 202 passes the events, one at a time, via 203 to the service logic 201.

[0028] Event communication supports decoupling of the service logic 201 and the user interfaces 204, and allows service providers to provide interchangeable user interfaces 204, or add new ones, to a single consistent source of service logic 201 and data.

[0029] In each round of interaction, the SISL infrastructure automatically sends out a set of events via 203 from the service logic 201 to the service monitor 202, corresponding to the events that are currently enabled in the service logic 201. There are three kinds of events: prompt events, up events, and notify events. Prompt events indicate to the user interface what information to communicate to the user and what information the service is ready to accept. There are three kinds of prompt events:

- *prompt_choice* events are disjunctive choices currently enabled in the service logic 201. For example, after the user has successfully logged into the Any-Time Teller banking service, a choice among the different transaction types is enabled. The service logic 201 sends a *prompt_choice_deposit*, *prompt_choice_withdrawal event*, and a *prompt_choice_transfer* event, and so forth, via 203 to the service monitor 202.

- *prompt_req* events are the events currently required by the service logic 201. For example, suppose the user has chosen to perform a transfer transaction. The Any-Time Teller requires that the user input a source account, a transfer account, and amount, and hence sends *prompt_req_src*, *prompt_req_tgt*, and *prompt_req_amt* via 203 to the service monitor 202.

- *prompt_opt* events are events enabled in the service logic 201, for which the user may correct previously given information. For example, suppose the user is performing a transfer and has already provided his/her source and target accounts, but not the amount. The service logic 201 sends *prompt_opt_src*, *prompt_opt_tgt*, and *prompt_req_amt* events via 203 to the service monitor 202. This indicates that the user may override the previously given source and target accounts with new information.

[0030] *Up* events correspond to prior points in the service logic 201 to which the user may go back. For example, the service logic 201 sends an *up_MainMenu* event via 203 to the service monitor 202. This allows the user to abort any transaction and go back up to the main menu.

[0031] *Notify* events are simply notifications that the user interface 204 should give the user; for example, that a transaction has completed successfully or that information provided by the user was incorrect or inconsistent.

[0032] Thus, the service monitor 202 is essentially a controllable interface that manages the communication between the service logic 201 and the multiple user interfaces 204. As such, service monitor 202 provides a communication mechanism in the form of a so-called "look ahead" in which the user can input information beyond what is currently required by the interactive service. The so-called look ahead is realized because the user interfaces 204 can supply information to service monitor 202 that is beyond, i.e., in addition, to that which is currently being requested by service logic 201. Thus, service monitor 202 may already have information from user interfaces 204 that may be later requested by, but is not currently being requested by, service logic 201. Note that service monitor 202 does not request any further information from user interfaces 204 until it does not have any information, i.e., input event, that matches an enabled event from service logic 201.

[0033] In summary, service monitor 202 is essentially a controllable memory unit including a bidirectional communication link 203 to service logic 201 and bidirectional communication links 205 to user interfaces 204. Service monitor 202 at any current instant stores a current set of user events from user interfaces 204, old user events not yet sent to service logic 201 and a current set of enabled events from service logic 201.

[0034] FIG. 3 is a flowchart illustrating the steps in implementing the operation of service monitor 202 in accordance with the invention. Accordingly, the process is started in step 301. Then, step 302 causes monitor 202 (FIG. 2) to wait for a set of enabled events from service logic 201. Step the causes a set of enabled events to be sent from service monitor 202 to user interfaces 204. Service monitor 202 is caused by step 304 to wait for a set of events corresponding to user inputs from user interfaces 204. Then, step 305 tests to determine whether any remaining user event matches some enabled event. If the test result in step 305 is NO, control is returned to step 303 and steps 303 through 305 are iterated until step 305 yields a YES result. Upon step 305 yielding a YES result, step 306 causes a matching user event to be selected in priority order, the removal of the selected matching user event from the set of user events, and sending the selected user event to service logic 201. Then, step 307 causes service monitor 202 to wait for a set of enabled events from service logic 201. Thereafter, control is returned to step 305 and, then, appropriate ones of steps 303 through 307 are iterated.

[0035] FIG. 4 is a flow diagram illustrating a reactive constraint graph for a portion of the Any-Time Teller banking service example.

[0036] In SISL, the service logic 201 (FIG. 2) of an application is specified as a *reactive constraint graph*, which is a directed acyclic graph with an enriched structure on the nodes. The traversal of reactive constraint graphs is driven by the reception of events from the environment; these events have an associated label, i.e., the event name, and may carry associated data. In response, the graph traverses its nodes and executes actions; the reaction of the graph ends when it needs to wait for the next event to be sent by the environment. For example, FIG. 4 shows a SISL reactive constraint graph that implements part of the functionality of the Any-Time Teller.

[0037] Reactive constraint graphs can have three kinds of nodes, namely, choice nodes, constraint nodes and action nodes.

[0038] Choice nodes represent a disjunction of information to be received from the user. Every choice node has a specified set of events. There are two forms of choice nodes, namely, event-based and data-based. Every event-based choice node has a specified set of events. For every event in this set, the SISL infrastructure automatically sends out a corresponding *prompt_choice* event from the service logic 201 to the user interface, via the service monitor 202. The choice node waits for the user interface to send, via the service monitor 202, any event in the specified set. When such an event arrives, the corresponding transition is taken, and control transfers to the descendent, i.e., child, node.

[0039] For example, if a *startTransfer* event arrives when control is at the choice node, the corresponding transition is taken and control is transferred to the target constraint node. To ensure determinism, all outgoing transitions of a choice node must be labeled with distinct event names.

[0040] Every data-based choice node has a specified set of preconditions on data. To ensure determinism, these preconditions must be specified so that exactly one of them is "true" in any state of the system. When control reaches a data-based choice node, the transition associated with the unique "true" precondition is taken, and control is transferred to the child node.

[0041] Constraint nodes represent a conjunction of information to be received from the user. Every constraint node has an associated set of constraints on events. Constraints have the following components:

- The *signature* is the set of events occurring in the constraint.
- The *evaluation function* is a boolean function on the events in the signature.
- The optional *satisfaction tuple* consists of an

optional action, not involving user interaction, and an optional notify function that may return a *notify* event with an associated message. If the constraint evaluates to true, the action is executed, the notify function is executed and the returned *notify* event is sent to the user interface via the service monitor 202.

- The optional *violation tuple* consists of an optional action, not involving user interaction, an optional notify function that may return a *notify* event with an associated message, an optional uplabel function that may return the uplabel of an ancestor node and an optional child node. If the constraint evaluates to "false", the action is executed, the notify function is executed and the returned *notify* event is sent to the user interfaces 204via the service monitor 202. The uplabel function is also executed; if it returns an ancestor's uplabel, it is generated, and hence control reverts back to the ancestor node. If no ancestor node's uplabel is returned and a child node is specified, control is transferred to the specified child node.

[0042] For example, amt<= Balance (src) of 407 in FIG. 4 is equivalent to ?amt<=Balance(?src), and is the evaluation of a prescribed constraint. The signature of this constraint is the set {*amt*, *src*}, and the satisfaction notify function and violation notify function, respectively, report to the user whether or not the source account has enough funds to cover the requested amount. The notification ?EventName refers to the data on the event with name EventName.

[0043] Every constraint node has the following components:

- An associated set of constraints. In the current semantics and implementation, this set is totally ordered, specifying the priority order in which the constraints are evaluated.

- An optional entry action, not involving user interaction.

- An optional finished tuple, consisting of an optional exit action, not involving user action, an optional notify function, an optional uplabel function and an optional child node.

[0044] The SISL infrastructure automatically sends out a *prompt_req* event, from the service logic 201 (FIG. 2) to the user interfaces 204via the service monitor 202, for every event that is still needed in order to evaluate some constraint. Additionally, the constraint node sends out a *prompt_opt* event for all other events mentioned in the constraints. These correspond to the information that can be corrected by the user.

[0045] In every round of interaction, the constraint node waits for the user interface to send, via the service monitor 202, any event that is mentioned in the signature of its associated constraints. Each constraint associated with a constraint node is evaluated as soon as all of its events have arrived. If an event is resent by the user interfaces, i.e., information is corrected, all constraints with that event in their signature are re-evaluated. For every evaluated constraint, its optional satisfied/violated action is automatically executed, and a *notify* event is automatically sent to the user interfaces, with the specified message.

[0046] Specifically, the constraints are evaluated in the specified priority order, currently the total ordered set. If any constraint is violated, the last received event is automatically erased from all constraints, since it caused an inconsistency. Furthermore, the violation action is executed, the notify function is executed and the returned *notify* event is automatically sent to the user interfaces 204via the service monitor 202. The uplabel function is also executed; if it returns an ancestor's uplabel, it is generated and, hence control reverts back to that ancestor. For example, in FIG. 4, the constraint node 403 for the transfer capability checks whether the source account is an active account of the given customer, i.e., user, via a prescribed constraint. If not, it generates the uplabel "LoginMenu", and control is transferred back to the Login node 401. Then, the user must re-enter his/her login. If no ancestor node's uplabel is returned and a child node is specified, control is transferred to that child node. For example, in FIG. 4, the Login node 401 checks whether the login is of a customer in good standing, via a constraint *good_customer*, which evaluates goodCustomer(*login*). If not, control is transferred to the Quit node and the service is terminated. If no ancestor node's uplabel is returned or child node specified for the violated constraint, the node reverts to waiting for events to arrive.

[0047] If no constraints have been violated, the action of every satisfied constraint is executed, the associated notify functions are executed and the returned *notify* events are sent to the user interfaces 204via the service monitor 202.

[0048] When all the constraints have been evaluated and are satisfied, the exit action and notify function associated with the constraint node are executed and the returned *notify* event is sent to the user interfaces 204via the service monitor 202. The uplabel function is also executed; if it returns an ancestor node's uplabel, it is generated, and hence control is returned back to the ancestor node. If no ancestor node's uplabel is returned and a child node it specified, control is transferred to that child node.

[0049] Action nodes represent some action, not involving user interaction, to be taken. After the action is executed, control transfers to the child node.

[0050] Nodes can have an optional "uplabel", which is used to transfer control from some child node back up to the node, allowing the user to revert back to previous points in the service. In each round of interaction, the

SISL infrastructure automatically sends out an *up* event, from the service logic 201 to the user interfaces 204via the service monitor 202, corresponding to the uplabel of every ancestor of the current node.

[0051] Nodes can also have a self-looping arc, with a boolean precondition on data. This indicates that the subgraph from the node will be repeatedly executed until the precondition becomes false.

[0052] An example execution of the Any-Time Teller Banking Service is shown in FIG. 4. The service initially causes the statement "Welcome to Any-Time Teller" to be spoken. The control point is at the root node, which is a constraint node. For the constraint of the root node to be satisfied, the *login* and *pin* values must be identified, i.e., *login==pin*, as shown in step 401 of FIG. 4. The SISL infrastructure automatically sends out a *prompt_req_login* and a *prompt req_pin* from the service logic 201 to the user interfaces, via the service monitor 202. The user interfaces, via the prompt function, respond by saying "Please specify your login and personal identification number". For the Web-based user interface, text fields for the login and pin are automatically generated, in known fashion; for the speech recognition-based user interface, the grammars specified in the grammar function are automatically enabled.

[0053] In this example, suppose that the user states "My login is Mary Smith and my pin is Mary Smith", and hence a *login* event with the value "Mary Smith" and a *pin* event with the value "Mary Smith" are sent to the service logic 201. Since the login and pin are identical, the constraint is satisfied. The SISL infrastructure automatically sends out a *notify* event with the message "Hello Mary Smith". Welcome to the SISL Any-Time Teller". The user interface makes this statement to the user.

• *Login successful*

[0054] Control now proceeds to step 402 and to the choice node. The SISL infrastructure automatically sends out:

- *prompt_choice_startDeposit*;
- *prompt_choice startWithdrawal*;
- *prompt_choice_startTransfer*; and
- *prompt_choice_userQuit*;

events from the service logic 201 (FIG. 2) to the user interfaces 204, via the service monitor 202, corresponding to the enabled choices. The user interfaces 204ask the user "What Transaction would you like to do?" For an automatic speech recognition-based user interface, if the user states "I need help", the user interface states, via a help function, "You can make a withdrawal, deposit transfer or you can quit the service". Consider that the user now chooses to perform a transfer, the *startTransfer* event is sent to the service logic 201.

• *Transfer*

[0055] Control now proceeds to constraint node 406. The SISL infrastructure automatically sends out:

- *prompt_req_src*;
- *prompt_req_tgt*; and
- *prompt _req_amt*

events from the service logic 201 to the user interfaces 204, via the service monitor 202, together with a *up_MainMenu* event, since it is the uplabel of an ancestor node. Assume that the user respond with "I would like to transfer One Thousand Dollars ($1,000.00)from my checking account", or equivalently "From checking, I would like to transfer One Thousand Dollars ($1,000.00)". Either order of the transfer request information is allowed; furthermore, this information in partial, since the target account is not specified. The user interface 204 sends a *src* event and an *amt* event, with the corresponding data, to the service monitor 202, which sends them one at a time to the service logic 201. Assume that the *src event is sent first*, followed by the *amt* event. The constraints *amt>o*, *isValid(src)* and *amt<=Balance (src)* are automatically evaluated.

[0056] Assume that the checking account does not have a balance of at least $1,000.00; hence, there is a constraint violation and the supplied information is erased, since it was sent last. Note that constraints are evaluated as soon as possible; for example, the user is not required to specify a target account in order for the balance on the source account to be checked. The SISL infrastructure then automatically sends out a *prompt_opt_src*, *prompt_req_tgt*, *prompt_req_amt* and *upMainmemu* events from the service logic 201 to the user interfaces, via the service monitor 202, as well as, a *notify* event with the message "Your checking account does not have sufficient funds to cover the amount of $1,000.00. Please specify an amount and a target account." The user interface 204 then notifies the user with this message and prompts the user for the information.

[0057] Assume now that the user states "Transfer Five hundred Dollars ($500.00) to savings". The *amt* and *tgt* events are sent to the service monitor 202, and passed to the service logic 201. The constraints are now all evaluated and satisfied, the service logic 201 automatically sends a *notify* event to the user interfaces 204 with the message. "Your transfer of $500.00 from checking to savings was successful".

[0058] Control then is returned back up to the choice node 402; the loop on the incoming arc to the choice node indicates that the corresponding subgraph is repeatedly executed until the condition on the arc becomes false. If the user wants to quit the service, the *userQuit* event is sent to the service logic 201, the *hasQuit* variable is set to true, and the loop is terminated. While in step 402, a loop step 404 is performed

to test if the user has terminated, i.e., quit, the session.

• *Back up to the Main Menu*

[0059] If the user would like to abort at any time during a withdrawal, deposit or transfer transaction, he/she can state "I would like to go back up to the Main Menu", which results in an *up_MainMenu* event to be sent to the service logic 201. This causes control to be returned to the choice node 402, which has an associated upMainMenu label.

• *Make a Deposit*

[0060] If a user wishes to make a deposit control proceeds to the deposit constraint node 406. The SISL infrastructure automatically sends out

- *prompt_req_tgt*,
- *prompt _req_amt*

events from the service logic 201 to the user interfaces 204 via service monitor 202. If the target account is valid and the amount is greater than zero (0) the deposit is made and the associated notification is executed.

• *Make a Withdrawal*

[0061] If a user wishes to make a withdrawal control proceeds to the deposit constraint node 407. The SISL infrastructure automatically sends out:

- *prompt_req_src*,
- *prompt_req_amt*

events from the service logic 201 to the user interfaces 204 via service monitor 202. If the source account is valid and the amount is greater than zero (0), it is determined if amt<=Balance(src) and, if so, the withdrawal is made and the associated notification is executed.

[0062] FIG. 5 illustrates the steps performed in an example execution of the Any-Time Teller including look ahead in accordance with the invention.

The service is started in an *INITIAL STATE OF SERVICE*. Then, in this example, the following steps are effected:

- SERVICE LOGIC 201: sends to Service Monitor 202 a set of enabled events. *prompt_req* events: {login, pin}.
- SERVICE MONITOR 202: sends set of enabled events to User Interfaces 204.
- USER INTERFACES 204: prompt the user with "Welcome to the Any-Time Teller banking service. Please specify your login and personal identification number".
- USER: states "I want to transfer money from my checking account. My login is JohnDoe and my pin is 4341".
- USER INTERFACES 204: send to Service Monitor 202 the set of user events: {startTransfer, src(checking), login(JohnDoe), pin(4341).
- MATCHING EVENTS: {login, pin}.
- SERVICE MONITOR: sends login(JohnDoe) event to Service Logic.
- SERVICE LOGIC 201: sends to Service Monitor 202 set of enabled events. *prompt_req* events: {pin}, *prompt_opt* events: {login}.
- REMAINING USER EVENTS: {startTransfer, src(checking), pin(4341)}.
- MATCHING EVENTS: {pin}.
- SERVICE MONITOR 202: send pin(4341) event to Service Logic 201.
- SERVICE LOGIC 201: sends to Service Monitor 202 set of enabled events. *prompt_choice* events: {startDeposit, startWithdrawal, startTransfer, userQuit}.
- REMAINING USER EVENTS: {startTransfer, src(checking)}.
- MATCHING EVENTS: {startTransfer}.
- SERVICE MONITOR 202: sends startTransfer event to Service Logic 201.
- SERVICE LOGIC 201: sends to Service monitor 202 set of enabled events. *prompt_req* events: {src, tgt, amt}.
- REMAINING USER EVENTS: {src(checking)}.
- MATCHING EVENTS: {src}.
- SERVICE MONITOR 202: sends src(checking) event to Service Logic 201.
- SERVICE LOGIC 201: sends to Service Monitor 202 a set of enabled events. *prompt_req* events: {tgt, amt}. *prompt_opt* events: {src}
- SERVICE MONITOR 202: sends set of enabled events to User Interfaces 204.
- USER INTERFACES 204: prompt user with "Hello JohnDoe. Please specify the target account and the

amount you would like to transfer".

[0063] The above-described embodiment is, of course, merely illustrative of the principles of the invention. Indeed, numerous other methods or apparatus may be devised by those skilled in the art without departing from the spirit and scope of the invention.

Claims

1. Apparatus for use in providing an interactive user service including,

   a service logic unit, and
   a plurality of user interfaces,
   the apparatus being CHARACTERIZED BY
   a service monitor unit for controlling communication of information between said service logic unit and a plurality of user interfaces, at least one of said user interfaces having natural language understanding, and said service monitor unit including
   receiver means adapted to receive information from any of said plurality of user interfaces and adapted to receive requests for information from said service logic unit, and
   correlator means adapted to match received information from said plurality of user interfaces to requested information from said service logic unit and to supply the matched received information to said service logic unit in a prescribed manner.

2. The apparatus as defined in claim 1 CHARACTERIZED IN THAT said service logic unit provides data and logic to said plurality of user interfaces and includes service logic means adapted to facilitate at least different ordering of inputs from any of said user interfaces, and/or incomplete information from any of said user interfaces, and/or correction of information from any of said user interfaces, and/or returning of control to prior points in said interactive service in response to a request from at least one of said plurality of user interfaces.

3. The apparatus as defined in claim 1 and 2 CHARACTERIZED IN THAT each of said user interfaces includes means adapted to allow users to supply information to said service monitor unit in addition to information currently requested by said service logic unit.

4. The apparatus as defined in claims 1 and 2 CHARACTERIZED IN THAT said communication of information is of events.

5. The apparatus as defined in claim 4 CHARACTERIZED IN THAT said requested information is a set of enabled events.

6. The invention as defined in claim 5 CHARACTERIZED IN THAT said service monitor unit includes means adapted to receive a set of input events from at least one of said user interfaces and means adapted to controllably supply said received input events in a prescribed order to said service logic unit.

7. The invention as defined in claim 6 CHARACTERIZED IN THAT said service logic includes means adapted to controllably generates a set of enabled events in response to said supplied events and means adapted to supply said set of enabled events to said service monitor unit, and said service monitor unit includes means adapted to supply said set of enabled events in a prescribed fashion to all appropriate ones of said user interfaces.

8. The apparatus as defined in claim 5 CHARACTERIZED IN THAT said information, received by said receiver means is a set of input events, and said correlator means includes means adapted to match said received input events to said enabled events and means is adapted to supply said matched input events one at a time to said service logic unit in a prescribed order.

9. The apparatus as defined in claim 8 CHARACTERIZED IN THAT said prescribed order is a prescribed priority order.

10. The apparatus as defined in claim 9 CHARACTERIZED IN THAT said priority order is prescribed by the service provider.

11. The apparatus as defined in claim 10 CHARACTERIZED IN THAT said service logic unit includes means adapted to supply a new set of enabled events to said service monitor unit in response to an input event from said service monitor unit, and said service monitor unit includes means adapted to notify said user interfaces to send a new set of events.

12. The invention as defined in claim 11 CHARACTERIZED IN THAT said service monitor unit includes transmitter means adapted to transmit said notification to said user interfaces instruction said user interfaces to send a new set of input events to said service monitor unit only when said correlator means indicates that no current input event matches an enabled event.

13. The invention as defined in claim 12 CHARACTERIZED IN THAT said transmitter means is adapted to transmit said set of enabled events to said user

interfaces from said service monitor unit only when said correlator means indicates that no current input event matches an enabled event.

14. A method for providing an interactive user service comprising a service logic unit, a service monitor unit and a plurality of user interfaces including the step of,

controlling communication of information through said service monitor unit between said service logic unit and said plurality of user interfaces,
the method being CHARACTERIZED IN THAT at least one of said user interfaces has natural language understanding, and said step of controlling communication includes the steps of
receiving information from any of said plurality of user interfaces,
receiving requests for information from said service logic unit,
matching received information to requested information, and
supplying the matched received information to said service logic unit in a prescribed manner.

15. The method as defined in claim 14 CHARACTERIZED BY providing data and logic from said service logic unit to said plurality of user interfaces and facilitating at least different ordering of inputs from any of said user interfaces, and/or incomplete information from any of said user interfaces, and/or correction of information from any of said user interfaces, and/or returning of control to prior points in said interactive service in response to a request from at least one of said plurality of user interfaces.

16. The method as defined in claims 14 and 15 CHARACTERIZED BY a step of supplying information from users of said user interfaces to said service monitor unit in addition to information currently requested by said service logic unit.

17. The method as defined in claims 14 and 15 CHARACTERIZED IN THAT said communication of information is of events.

18. The method as defined in claim 17 CHARACTERIZED IN THAT said requested information is a set of enabled events.

19. The method as defined in claim 18 CHARACTERIZED BY the steps of receiving a set of input events from at least one of said user interfaces and controllably supplying said received input events in a prescribed order to said service logic unit.

20. The method as defined in claim 19 CHARACTERIZED BY the steps of controllably generating a set of enabled events in response to said supplied input events, supplying said set of enabled events to said service monitor unit, and supplying said set of enabled events from said service monitor unit in a prescribed fashion to all appropriate ones of said user interfaces.

21. The method as defined in claim 18 CHARACTERIZED IN THAT said received information is a set of input events, and said step of matching includes matching said received input events to said enabled events and said step of supplying supplies said matched input events one at a time to said service logic unit in a prescribed order.

22. The method as defined in claim 21 CHARACTERIZED IN THAT said prescribed order is a prescribed priority order.

23. The method as defined in claim 22 CHARACTERIZED IN THAT said priority order is prescribed by the service provider.

24. The method as defined in claim 23 CHARACTERIZED BY a step of notifying said user interfaces to send a new set of events.

25. The method as defined in claim 24 CHARACTERIZED IN THAT said step of notifying transmits a notification to said user interfaces to transmit a new set of input events to said service monitor unit only when said step of matching indicates that no current input event matches an enabled event.

26. The method as defined in claim 25 CHARACTERIZED BY a step of transmitting said set of enabled events to said user interfaces from said service monitor unit only when said step of matching indicates that no current input event matches an enabled event.

FIG. 1

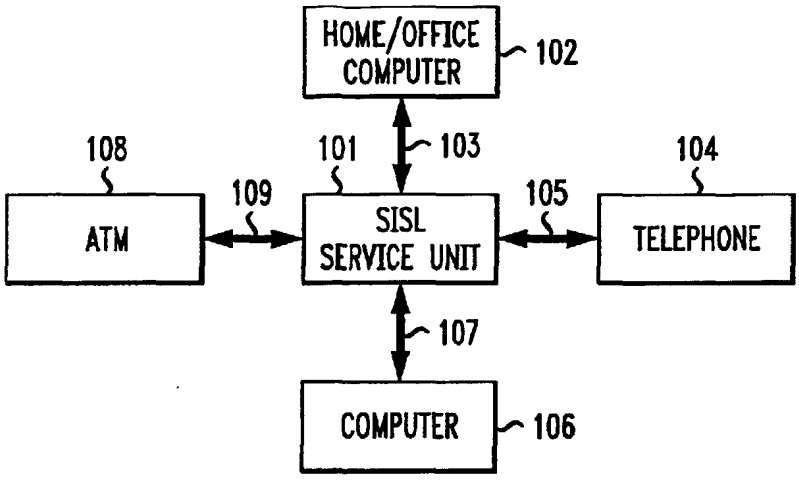

HOME/OFFICE COMPUTER
102
108
101
103
104
109
105
ATM
SISL SERVICE UNIT
TELEPHONE
107
COMPUTER
106

FIG. 2

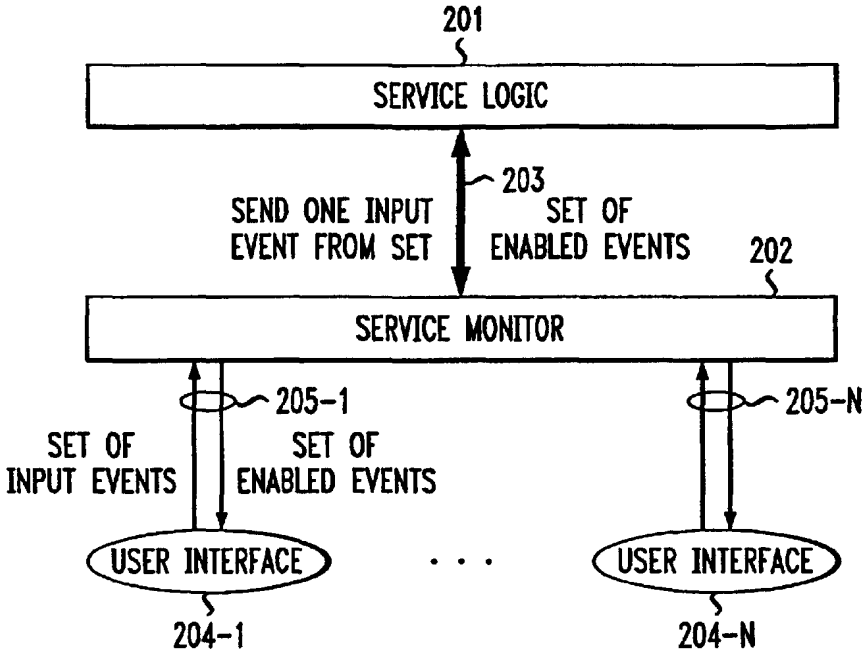

201
SERVICE LOGIC
203
SEND ONE INPUT EVENT FROM SET
SET OF ENABLED EVENTS
202
SERVICE MONITOR
205-1
205-N
SET OF INPUT EVENTS
SET OF ENABLED EVENTS
USER INTERFACE
USER INTERFACE
204-1
204-N

FIG. 3

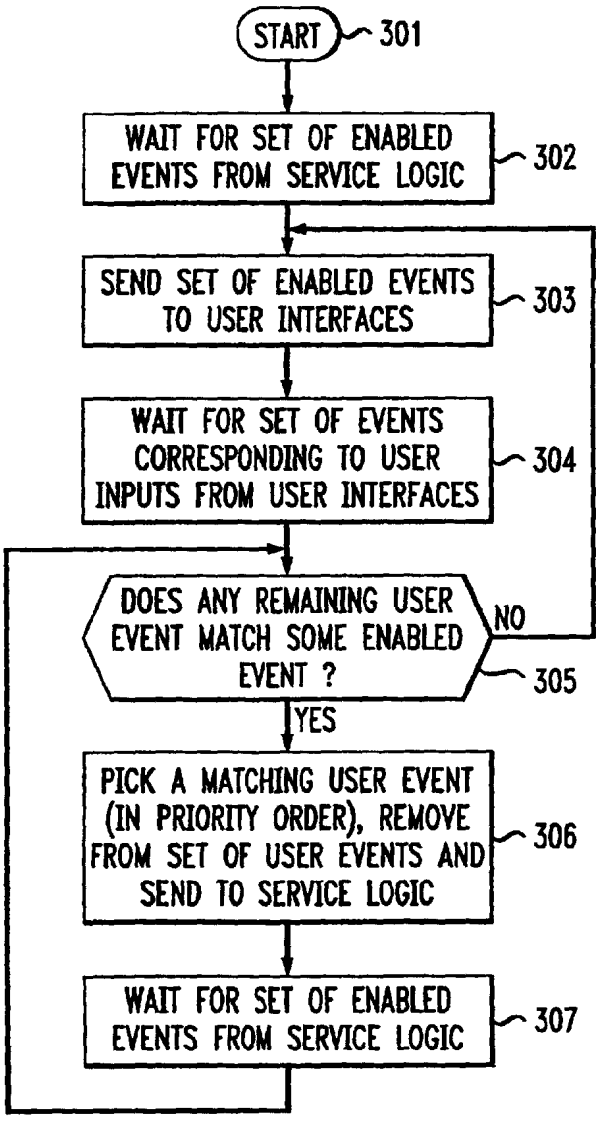

START
301
WAIT FOR SET OF ENABLED EVENTS FROM SERVICE LOGIC
302
SEND SET OF ENABLED EVENTS TO USER INTERFACES
303
WAIT FOR SET OF EVENTS CORRESPONDING TO USER INPUTS FROM USER INTERFACES
304
DOES ANY REMAINING USER EVENT MATCH SOME ENABLED EVENT ?
NO
305
YES
PICK A MATCHING USER EVENT (IN PRIORITY ORDER), REMOVE FROM SET OF USER EVENTS AND SEND TO SERVICE LOGIC
306
WAIT FOR SET OF ENABLED EVENTS FROM SERVICE LOGIC
307

FIG. 4

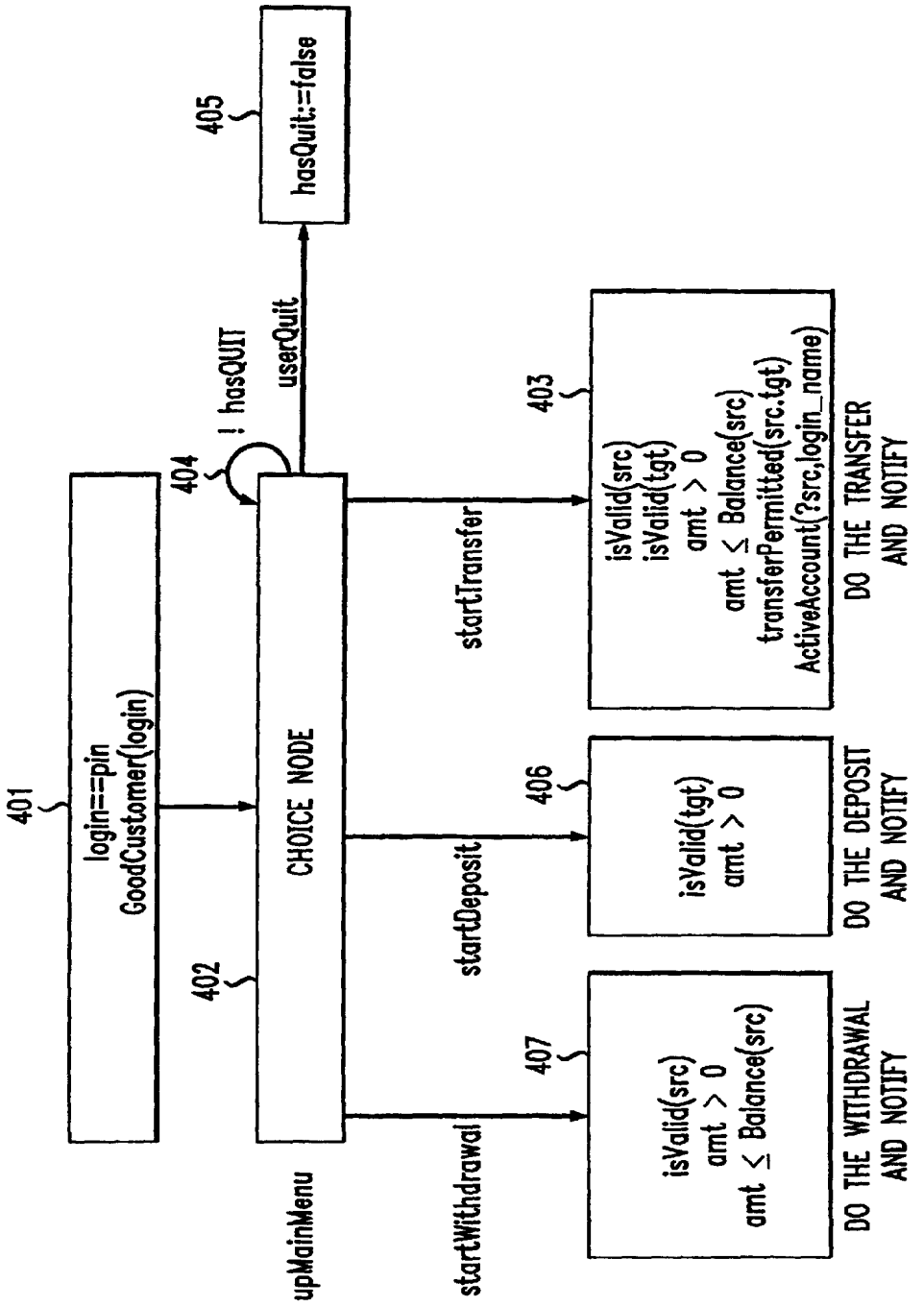

401
login==pin
GoodCustomer(login)
402
CHOICE NODE
404
! hasQUIT
userQuit
405
hasQuit:=false
403
startTransfer
isValid(src)
isValid(tgt)
amt > 0
amt ≤ Balance(src)
transferPermitted(src,tgt)
ActiveAccount(?src,login_name)
DO THE TRANSFER
AND NOTIFY
406
startDeposit
isValid(tgt)
amt > 0
DO THE DEPOSIT
AND NOTIFY
407
startWithdrawal
isValid(src)
amt > 0
amt ≤ Balance(src)
DO THE WITHDRAWAL
AND NOTIFY
upMainMenu

FIG. 5

INITIAL STATE OF SERVICE

SERVICE LOGIC: SENDS TO SERVICE MONITOR SET OF ENABLED EVENTS.
prompt_req events: {login, pin}

SERVICE MONITOR: SENDS SET OF ENABLED EVENTS TO USER INTERFACES

USER INTERFACES: PROMPT THE USER WITH "WELCOME TO THE ANY-TIME TELLER BANKING SERVICE. PLEASE SPECIFY YOUR LOGIN AND PERSONAL IDENTIFICATION NUMBER".

USER: STATES "I WANT TO TRANSFER MONEY FROM MY CHECKING ACCOUNT.
MY LOGIN IS JohnDoe AND MY PIN IS 4341"

USER INTERFACES: SEND TO SERVICE MONITOR THE SET OF USER EVENTS:
{startTransfer, src(checking), login(JohnDoe), pin(4341)

MATCHING EVENTS: {login, pin}

SERVICE MONITOR: SENDS login(JohnDoe) EVENT TO SERVICE LOGIC

SERVICE LOGIC: SENDS TO SERVICE MONITOR SET OF ENABLED EVENTS.
prompt_req events: {pin}, prompt_opt events: {login}

REMAINING USER EVENTS: {startTransfer, src(checking), pin(4341)}

MATCHING EVENTS: {pin}

SERVICE MONITOR: SEND pin(4341) EVENT TO SERVICE LOGIC

SERVICE LOGIC: SENDS TO SERVICE MONITOR SET OF ENABLED EVENTS.
prompt_choice events: {startDeposit, startWithdrawal, startTransfer, userQuit}

REMAINING USER EVENTS: {startTransfer, src(checking)}

MATCHING EVENTS: {startTransfer}

SERVICE MONITOR: SENDS START TRANSFER EVENT TO SERVICE LOGIC

SERVICE LOGIC: SENDS TO SERVICE MONITOR SET OF ENABLED EVENTS.
prompt_req events: {src, tgt, amt}

REMAINING USER EVENTS: {src(checking)}

MATCHING EVENTS: {src}

SERVICE MONITOR: SENDS src(checking) EVENT TO SERVICE LOGIC

SERVICE LOGIC: SENDS TO SERVICE MONITOR SET OF ENABLED EVENTS.
prompt_req events: {tgt, amt}, prompt_opt events: {src}

SERVICE MONITOR: SENDS SET OF ENABLED EVENTS TO USER INTERFACES

USER INTERFACES: PROMPT THE USER WITH "HELLO JohnDoe. PLEASE SPECIFY THE TARGET ACCOUNT AND THE AMOUNT YOU WOULD LIKE TO TRANSFER".